Softwaretechnik

Lecture 19: Model Driven Engineering

Peter Thiemann

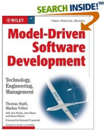
University of Freiburg, Germany

23.07.2012

Introduction MDA

Material

- Thomas Stahl, Markus Völter. Model-Driven Software Development. Wiley & Sons. 2006.



- Anneke Kleppe, Jos Warmer. MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson. 2003.
- Stephen J. Mellor, Axel Uhl, Kendall Scott, Dirk Weise. MDA Distilled: Solving the Integration Problem with the Model Driven Architecture. Pearson. 2004.

What is MDA?

- MDA = Model Driven Architecture
  - also: MD (Software/Application) Development, Model Based [Development/Management/Programming]
  - Model Driven Engineering, Model Integrated Computing
- Initiative of the OMG (trade mark)
  - OMG = Object Management Group: CORBA, UML, ...
  - open consortium of companies (ca. 800 Firmen)
- Goal: Improvement of software development process
  - Interoperability
  - Portability
- Approach: Shift development process from code-centric to model-centric
  - Reuse of models
  - Transformation of models
  - Code generation from models

Goals of MDA

Higher Degree of Abstraction

Portability and Reusability

- Development abstracts from target platform
- Technology mapping in reusable transformations
- New technology $\Rightarrow$ new transformation

Interoperability

- Systems span several platforms
- Information flow between platforms via *bridges*
- Byproduct of model transformations

Goals of MDA

Models and Model Transformations

Productivity

Every development phase directly contributes to the product, not just the implementation

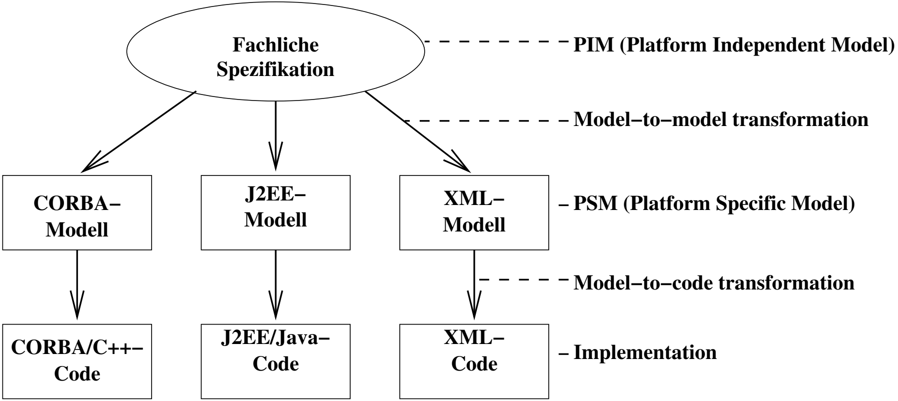
Documentation and Maintenance

- Changes through changes of the models
- Models are documentation $\Rightarrow$ consistency
- Separation of concern
- Better handle on changing technology

Specialization

- Business processes
- Technologies

Models in MDA



Models in MDA/2

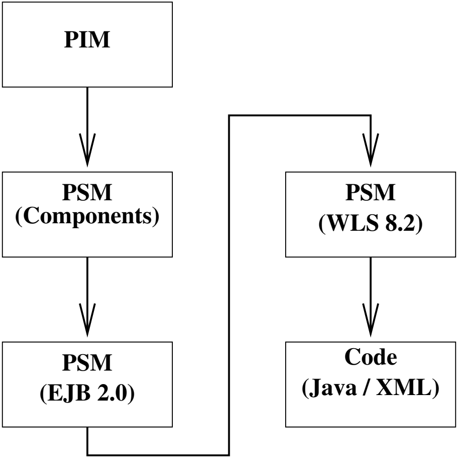
PIM vs PSM

- Relative concepts
- Smooth transition
- Several levels of model and transformation steps possible
- Inverse transformation PSM $\Rightarrow$ PIM unlikely

Transformation

- Code is the ultimate model (PSM)
- Model-to-code is a special case

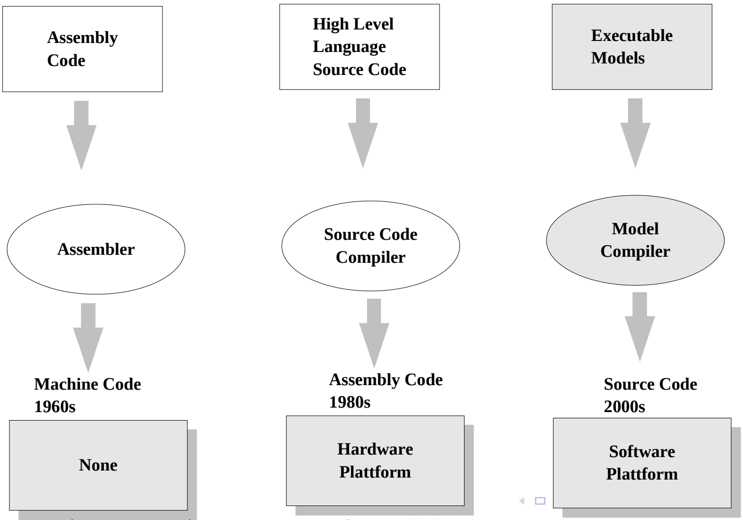
Models and Transformations



Platform

- API
- Virtual machine
- Provides several services
- Examples
  - Different processors ⇒ hardware platform
  - Operating system ⇒ software platform
  - Java VM ⇒ software platform
  - EJB ⇒ component platform
  - CORBA, Webservices, ...
  - Application architecture, DSL (Domain Specific Language)

Examples for Platforms



Transformations

- Mappings between models
- Formal definition required for automatic application
- Standardized transformation language QVT (Queried Views and Transformations)
  Distilled from 23 very different proposals
- Tools
  - Transformations based on metamodel
  - Code generator via patterns
  - Proprietary transformation languages (scripting)

Metamodeling

Metamodeling

Intro

- What?
  - meta = above
  - Define an ontology of concepts for a domain.
  - Define the **vocabulary** and **grammatical rules** of a modeling language.
  - Define a domain specific language (DSL).
- Why?
  - Concise means of specifying the set models for a domain.
  - Precise definition of modeling language.
- How?
  - Grammars and attributions for textbased languages.
  - Metamodeling generalizes to arbitrary languages (*e.g.*, graphical)

Metamodeling

Uses

- Construction of DSLs
- Validation of Models
  (checking against metamodel)
- Model-to-model transformation
  (defined in terms of the metamodels)
- Model-to-code transformation
- Tool integration

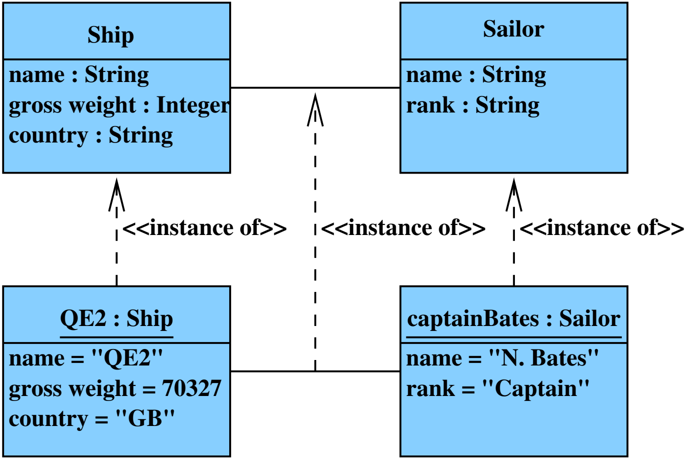
Excursion: Classifiers and Instances

- Classifier diagrams may also contain instances
- Instance description may include
  - name (optional)
  - classification by zero or more classifiers
  - kind of instance
    - instance of class: object
    - instance of association: link
    - etc
  - optional specification of values

Excursion: Notation for Instances

- Instances use the same notation as classifier
  - Box to indicate the instance
  - Name compartment contains
    <u>*name*: *classifier*, *classifier*...</u>
    <u>*name*: *classifier*</u>
    <u>: *classifier*</u> anonymous instance
    <u>:</u> unclassified, anonymous instance
  - Attribute in the classifier may give rise to like-named **slot** with optional value
  - Association with the classifier may give rise to **link** to other association end
    direction must coincide with navigability

Excursion: Notation for Instances (Graphical)



Terminology/Syntax

well-formedness rules

- abstract syntax
  just structure, how are the language concepts composed
- concrete syntax
  defines specific notation
- typical use:
  parser maps concrete syntax to abstract syntax

Terms/Abstract Syntax

Example: Arithmetic expressions

- abstract syntax

```
data Expr = Const String
          | Var String
          | Binop Op Expr Expr
data Op   = Add | Sub | Mul | Div

Binop Mul (Const "2")
          (Binop Add (Var "x") (Const "3"))
```

- concrete syntax

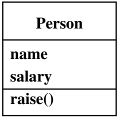
$$
\begin{array}{lcl}
E & ::= & \mathtt{c} \mid \mathtt{x} \mid E\ B\ E \mid (E) \\
B & ::= & + \mid - \mid * \mid /
\end{array}
$$

```
2 * (x + 3)
```

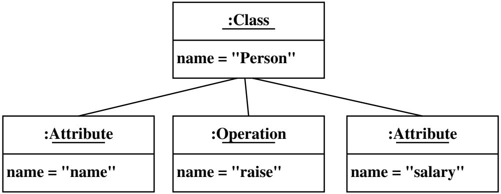
Terms/Abstract Syntax

Example: UML class diagram

- concrete syntax



- abstract syntax



Terms/Static Semantics

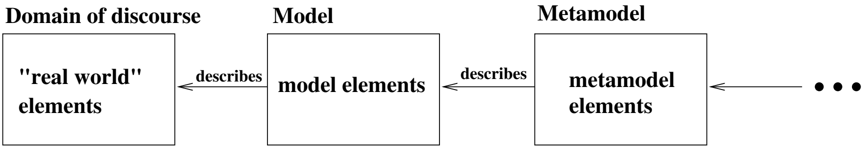
- **Static semantics** defines well-formedness rules beyond the syntax
- Examples
  - "Variables have to be defined before use"
  - Type system of a programming language
    `"hello" * 4` is syntactically correct Java, but rejected
- UML: static semantics via OCL expressions
- Use: detection of modeling/transformation errors

Terms/Domain Specific Language (DSL)

- Purpose: formal expression of key aspects of a domain
- Metamodel of DSL defines abstract syntax and static semantics
- Additionally:
  - concrete syntax (close to domain)
  - dynamic semantics
    - for understanding
    - for automatic tools
- Different degrees of complexity possible
  configuration options with validity check
  graphical DSL with domain specific editor

Model and Metamodel

Model and Metamodel

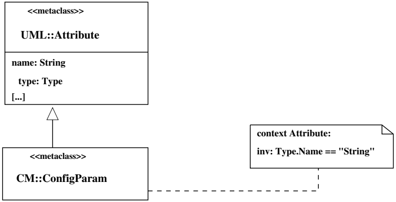


- Insight: **Every model is an instance of a metamodel.**
- Essential: *instance-of* relationship
- Every element must have a classifying metaelement which
  - contains the metadata and
  - is accessible from the element
- Relation Model:Metamodel is like Object:Class
- Definition of Metamodel by Meta-metamodel
- $\Rightarrow$ infinite tower of metamodels
- $\Rightarrow$ "meta" relation always relative to a model

Metamodeling a la OMG

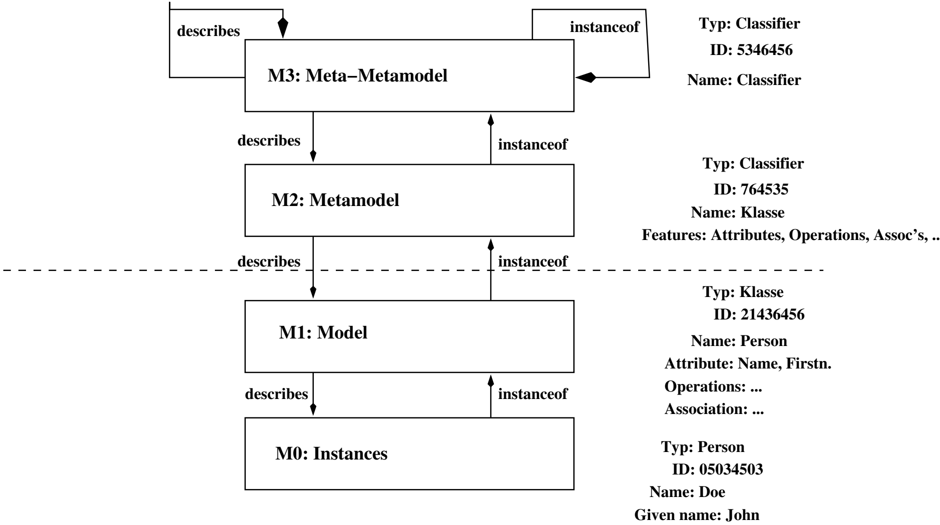
- OMG defines a standard (MOF) for metamodeling
- MOF (Meta Object Facilities) used for defining UML
- Confusion alert:
  - MOF and UML share syntax (classifier and instance diagrams)
  - MOF shares names of modeling elements with UML (*e.g.*, Class)
- Approach
  - Restrict infinite number of metalevels to **four**
  - Last level is deemed "self-describing"

Metamodeling and OCL



- OCL constraints are independent of the modeling language and the metalevel
- OCL on layer M$n + 1$ restricts instances on layer Mn

OMG's Four Metalevels



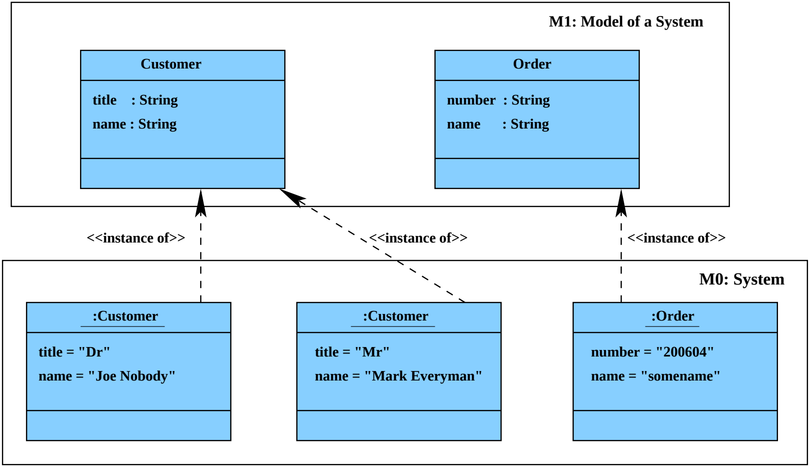
Layer M0: Instances

- Level of the running system
- Contains actual objects, *e.g.*, customers, seminars, bank accounts, with filled slots for attributes etc
- Example: object diagram

Layer M1: Model

- Level of system models
- Example:
  - UML model of a software system
  - Class diagram contains modeling elements: classes, attributes, operations, associations, generalizations, . . .
- Elements of M1 categorize elements at layer M0
- Each element of M0 is an instance of M1 element
- No other instances are allowed at layer M0

Relation between M0 and M1

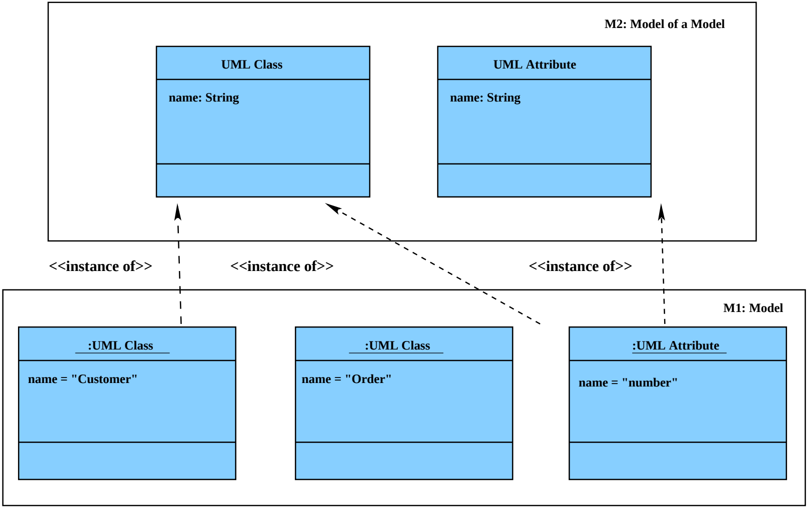


Layer M2: Metamodel

"Model of Model"

- Level of modeling element definition
- Concepts of M2 categorize instances at layer M1
- Elements of M2 model **categorize** M1 elements: classes, attributes, operations, associations, generalizations, ...
- Examples
  - Each class in M1 is an instance of some class-describing element in layer M2 (in this case, a *Metaclass*)
  - Each association in M1 is an instance of some association-describing element in layer M2 (a *Metaassociation*)
  - and so on

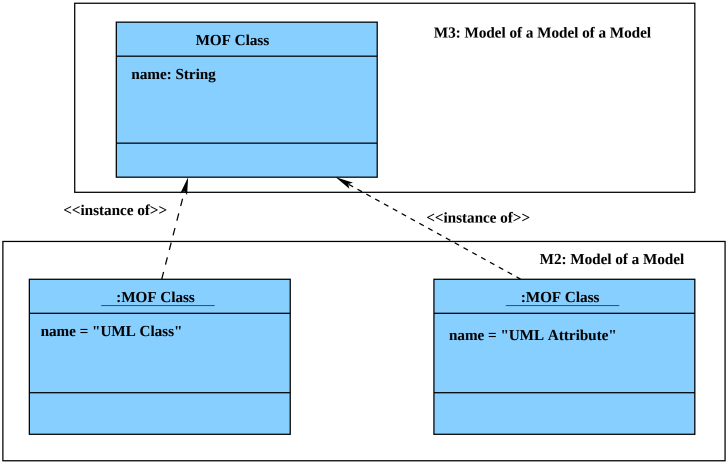
Relation between M1 and M2



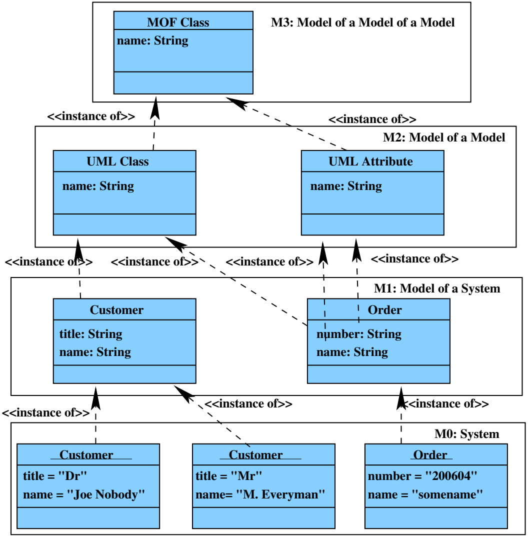
Layer M3: Meta-Metamodel

- Level for defining the definition of modeling elements
- Elements of M3 model **categorize** M2 elements: Metaclass, Metaassociation, Metaattribute, etc
- Typical element of M3 model: MOF class
- Examples
  - The metaclasses Class, Association, Attribute, etc are all instances of MOF class
- M3 layer is self-describing

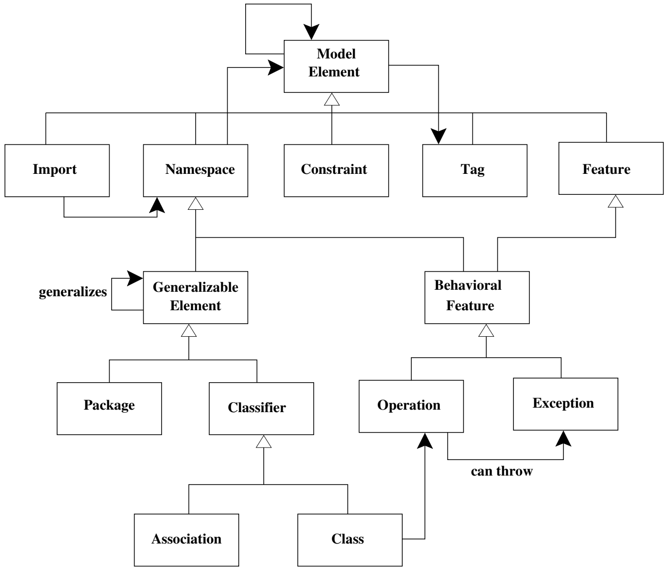
Relation between M2 and M3



Overview of Layers



Excerpt from MOF/UML

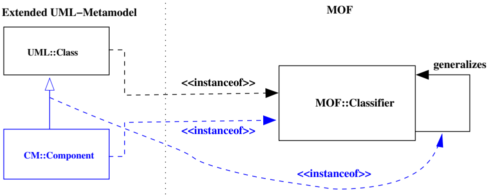


Extending UML
Designing a DSL

Designing a DSL

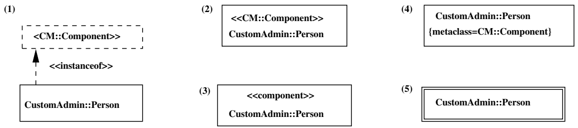
- Definition of a new M2 language from scratch too involved
- Typical approach: Extension of UML
- Extension Mechanisms
  - Extension of the UML 2 metamodel
    applicable to all MOF-defined metamodels
    (heavyweight)
  - Extension using **stereotypes** and **profiles**
    (lightweight)

Extending the UML Metamodel



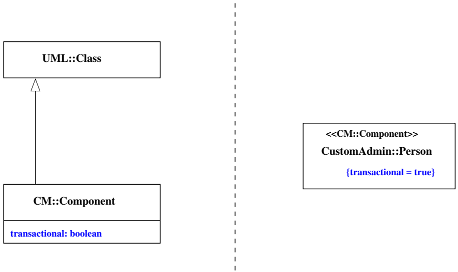
- MOF sanctions the derivation of a new metaclass **CM::Component** from **UML::Class**
- **CM::Component** is an instance of **MOF::Classifier**
- the generalization is an instance of MOF's **generalizes** association

Extending the UML Metamodel/Concrete Syntax



1. Explicit instance of metaclass
2. Name of metaclass as stereotype
3. Convention
4. Tagged value with metaclass
5. Own graphical representation (if supported)

Adding to a Class



- "just" inheriting from **UML::Class** leads to an identical copy
- Adding an attribute to the **CM::Component** metaclass leads to
  - an attribute value slot in each instance
  - notation: tagged value (typed in UML 2)

Stereotypes and Profiles

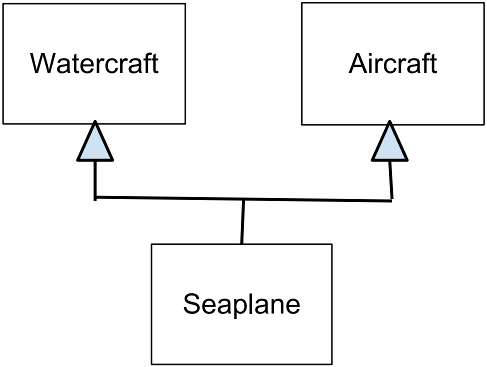
Stereotype

- Annotation to specialize UML elements
- Formally an extension of a metaclass
- Notation: Name in «Guillemets»

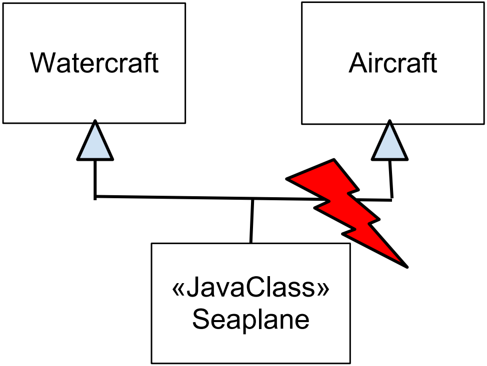
Profile

Package of stereotypes

Example



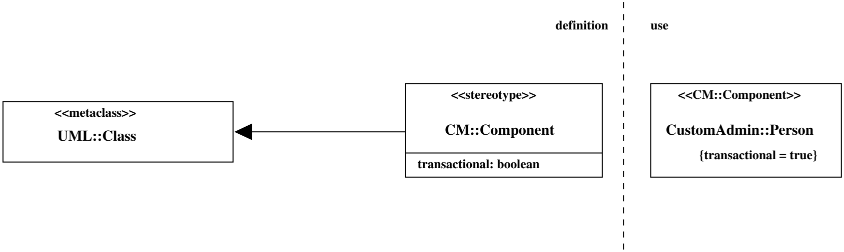
Example with Stereotype Added



Extension Using Stereotypes

- Simple specialization mechanism of UML
- No recourse to MOF required
- Tagged Values untyped
- No new metaassociations possible

Extending Using Profiles (UML 2)

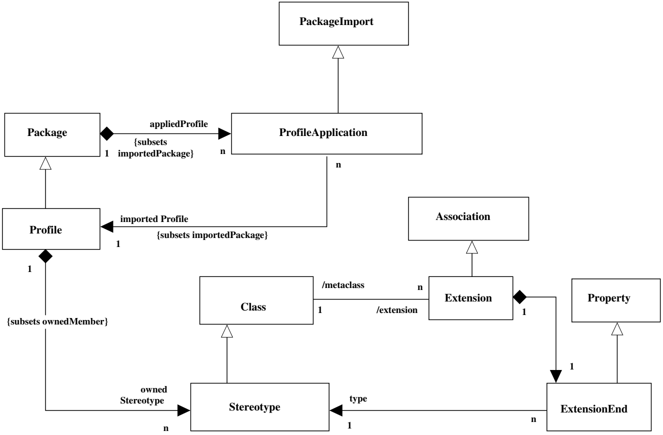


- Extension of the stereotype mechanism
- Requires "Extension arrow" as a **new UML language construct** (generalization with filled arrowhead)
- Not: generalization, implementation, stereotyped dependency, association, . . .
- Attributes ⇒ typed tagged values
- Multiple stereotypes possible

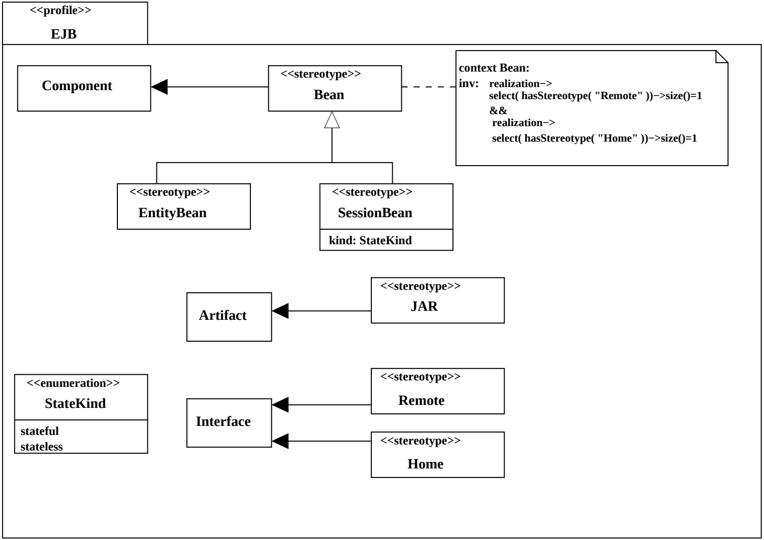
More on Profiles

- Profiles make UML into a **family of languages**
- Each member is defined by application of one or more profiles to the base UML metamodel
- Tools should be able to load profiles and corresponding transformations
- Profiles have three ingredients
  - stereotypes
  - tagges values
  - constraints
- Profiles can only impose further restrictions
- Profiles are formally defined through a metamodel

Profile Metamodel



Example Profile for EJB



Further Aspects of Profiles

- Stereotypes can inherit from other stereotypes
- Stereotypes may be abstract
- Constraints of a stereotype are enforced for the stereotyped classifier
- Profiles are relative to a reference metamodel
  e.g., the UML metamodel or an existing profile
- Most tools today do not enforce profile-based modeling restrictions, so why bother with profiles?
  - constraints for documentation
  - specialized UML tools
  - validation by transformer / program generator